IBM solidDB
IBM solidDB Universal Cache

V6.3





内存数据库用户指南

Information Management software

S151-1146-00

IBM solidDB
IBM solidDB Universal Cache

V6.3

IBM



内存数据库用户指南

Information Management software

S151-1146-00

声明

在使用本资料及其支持的产品之前，请阅读第 45 页的『声明』中的信息。

本版本适用于 IBM solidDB（产品编号 5724-V17）和 IBM solidDB Universal Cache（产品编号 5724-W91）V6R3 及所有后续发行版和修订版，直到在新版本中另有声明为止。

目录

图 v

表 vii

关于本手册 ix
印刷约定 ix
语法表示法约定 x

1 基本特性 1
内存表 1
内存表与基于磁盘的表 1
内存表的类型 1
内存表能够提高性能的原因 3
决定将哪些表指定为内存表的方法 3
用于指定一个表存储在内存中的方法 4
内存消耗量 4
计算磁盘空间需求 10
是否符合标准 10
内存表的局限性 11
其他内存引擎增强功能 11
将 solidDB 链接库访问和 HotStandby 与 solidDB 内存引擎配合使用 12
链接库访问 12
HotStandby 12
与先前的 solidDB 产品的不兼容性 12

2 临时表和瞬态表 13
临时表 13
有限的可视性 14
有限的持续时间 14
附加局限性 14
瞬态表 15
临时表与瞬态表之间的差别 16

3 优化和调整服务器 19
用于选择要存储在内存中的表的算法 19
临时表和瞬态表的性能调整信息 20
索引 21

4 配置内存数据库 23
配置文件和参数设置 23
管理服务器端的参数 24
使用 ADMIN COMMAND 查看和设置参数 . . . 24
在 solid.ini 中查看和设置参数 27
常量参数值 27

附录 A. 计算最大 BLOB 大小 29
用途 29
背景 29
计算 29

附录 B. 计算存储空间需求 33
基于磁盘的表 33
内存表 34
列大小表 34
测量内存消耗量 35
详细信息 35
基于磁盘的表 35
内存表 36

附录 C. 配置参数 39
[General] 节 39
[MME] 节 39

索引 43

声明 45


图

表

1. 印刷约定 ix
2. 语法表示法约定 x
3. 引用约束 17
4. 计算可用于 BLOB 数据的空间 30
5. 存储值所需要的字节数 30
6. 标题字节数 35
7. MME 参数 39
8. MME 参数 40

关于本手册

IBM® solidDB® 内存数据库提供了独特的双引擎数据库管理系统（DBMS）体系结构，从而使您能够在获得最佳性能与能够处理大量数据之间找到最佳平衡点。数据库服务器中有两个引擎：一个是主内存引擎（MME），用于以最快速度访问性能关键数据；另一个是“传统的”基于磁盘的引擎，用于高效处理任何容量的数据。

solidDB 主内存引擎构建于 solidDB 基于磁盘的引擎和 solidDB 功能，这意味着 solidDB 主内存引擎将拥有这些产品的所有功能。solidDB 主内存引擎可以用于嵌入式系统，实际上不需要进行管理或维护。可以通过将 solidDB 作为高可用性配置来部署，从而使 solidDB 主内存引擎适合于高可用性系统。还可以部署“高级复制”组件，它使得多个使用 solidDB 主内存引擎的服务器和使用 solidDB 基于磁盘的引擎的服务器能够互相共享数据和使数据同步。

本指南介绍了一些功能部件，借助这些功能部件并通过使用内存数据库技术就可以优化数据库服务器的性能。

本指南假定读者已经具备关系数据库管理系统的一般知识并且熟悉 SQL。本指南还假定读者基本上熟悉 solidDB 产品系列。建议您在阅读本指南之前阅读《IBM solidDB 管理员指南》。如果您尚不熟悉关系数据库，那么建议您首先阅读《IBM solidDB 入门指南》和《IBM solidDB SQL 指南》。

印刷约定

solidDB 文档使用下列印刷约定：

表 1. 印刷约定

格式	适用于
数据库表	此字体用于所有普通文本。
NOT NULL	采用此字体的大写字母指示 SQL 关键字和宏名称。
solid.ini	这些字体指示文件名和路径表达式。
`SET SYNC MASTER YES;` `COMMIT WORK;`	此字体用于程序代码和程序输出。示例 SQL 语句也使用此字体。
run.sh	此字体用于样本命令行。
TRIG_COUNT()	此字体用于函数名。
java.sql.Connection	此字体用于接口名称。
LockHashSize	此字体用于参数名、函数自变量和 Windows® 注册表条目。
argument	此类强调词指示用户或应用程序必须提供的信息。

表 1. 印刷约定（续)

格式	适用于
管理指南	这种样式用于引用其他文档或者同一文档中的章节。新术语和强调的问题也按此样式书写。
文件路径表示	文件路径按 UNIX® 格式提供。斜杠（/）字符表示安装根目录。
操作系统	如果文档包含有关操作系统之间的差别的内容，那么首先提到的是 UNIX 格式。Microsoft® Windows 格式位于 UNIX 格式之后并括在括号中。其他操作系统将单独列出。对于不同的操作系统还可能有不同的章节进行描述。

语法表示法约定

solidDB 文档使用下列语法表示法约定:

表 2. 语法表示法约定

格式	适用于
`INSERT INTO` *`table_name`*	语法描述采用此字体。可替换部分采用*此*字体。
solid.ini	此字体指示文件名和路径表达式。
[ ]	方括号指示可选项；如果是粗体文本，那么必须将方括号包含在语法中。
|	竖线，用于将语法行中的两个互斥选项分隔开。
{ }	大括号用于对语法行中的一组互斥选项进行定界；如果是粗体文本，那么必须将大括号包括在语法中。
...	省略号指示可以多次重复使用自变量。
. . .	由三个点组成的一列表示这是先前代码行的延续。

1 基本特性

IBM® solidDB 主内存引擎既具有内存表的高性能，又具有基于磁盘的表的容量几乎不受限制的特性。这是当前市面上的其他解决方案所不具备的优势。纯粹的内存数据库的运行速度很快，但是受到内存大小的严格限制。纯粹的基于磁盘的数据库允许存储器数量几乎不受限制，但是它们的性能受到磁盘访问速度的控制。即使计算机有足够的内存将整个数据库存储在内存缓冲区中，针对基于磁盘的表设计的数据库服务器的运行速度也比较慢，这是因为对于基于磁盘的表最佳的数据结构与对于内存表最佳的数据结构大不相同。

针对这种情况，solidDB 提出的解决方案是提供单个数据库服务器，而此数据库服务器中包含两个经过优化的服务器：一个服务器是针对基于磁盘的访问进行优化的，另一个服务器是针对内存访问进行优化的。这两个服务器共存于同一个进程中，使用一个 SQL 语句就可以访问两个引擎中的数据。

内存表

内存表与基于磁盘的表

如果一个表被指定为内存表，那么此表的所有内容都存储在内存中，以便能够尽可能快地访问数据。另一方面，如果一个表是基于磁盘的表，那么数据主要存储在磁盘上，并且服务器通常一次只将少量数据复制到内存中。

内存表与基于磁盘的表在大多数方面都相似。最重要的是，这两种表都完全持久保存数据（除非另有声明）。您可以对每种表都执行相同类型的查询。可以将基于磁盘的表和内存表组合在同一个 SQL 查询中。还可以将内存表和基于磁盘的表组合在同一事务中。使用 SA API（这是链接库访问的一部分）时，可以同时查询这两种表。此外，内存表可与索引、触发器和存储过程等配合使用。尽管对于非持久表的外键约束有一些限制，但是内存表还允许添加约束（其中包括主键约束和外键约束）。（后面更详细地讨论了此主题。）

借助 solidDB，用户可以决定哪些表将是内存表，哪些表将是基于磁盘的表。您可以将大量使用的表放入主内存中，以便更快地访问这些表。如果您有足够的内存，那么可以将所有表都放入主内存中。

内存表的类型

solidDB 提供了两种不同类别的内存表：持久内存表和非持久内存表。后续章节中对它们进行了描述。

持久内存表

顾名思义，将无限期地持久保存持久内存表。尽管客户机查询将访问数据在内存中的副本，但是当服务器关闭时会将内存表存储在磁盘上。因此，每当服务器启动时，内存表中的数据仍然可用。内存表还使用事务记录，以便在服务器意外关闭时（例如，因电源故障而关闭），服务器将记录已发生的事务，并且可以更新这些表以确保它们

具有所有已落实事务的数据。与基于磁盘的表一样，在执行检查点操作期间，内存表会将它们的数据复制到硬盘驱动器中（有关执行检查点操作的描述，请参阅《IBM solidDB 管理员指南》）。

还可以将内存表与 solidDB 的 HotStandby 组件配合使用；当主服务器发生故障时，会将内存表中的数据从它所在位置复制到辅助服务器中。

在设计使用内存表的应用程序时，应当考虑内存表与基于磁盘的表之间的一些差别。最重要的是，在缺省情况下，内存表始终使用悲观的行级别并行控制（锁定），而基于磁盘的表将使用乐观（版本控制）并行控制。因此，在处理读取事务期间，内存表读操作将阻止写操作。而且，内存表有可能发生死锁，而采用了版本控制的基于磁盘的表不会发生死锁。另一方面，当使用乐观并行控制时，可能会发生并行性冲突。

在设计应用程序的错误处理时，请务必记住这些注意事项。根据您使用的表类型不同，错误处理需要考虑不同的错误代码。

它们之间的另一个重要差别是使用了执行检查点操作算法。内存表的执行检查点操作算法与基于磁盘的表使用的算法完全不同。这种算法的主要好处是执行检查点操作期间不会以任何方式阻止事务访问这些表。因此，与基于磁盘的表相比，可以更准确地预测内存表的响应时间。

内存表与基于磁盘的表之间的第三个主要差别是，对于内存表，决不会将辅助索引写入磁盘中。只是将它们维护在内存中，当服务器启动时将重新构建索引。因此，与基于磁盘的表相比，它们对写入性能的影响小得多。此外，内存表的所有索引的速度相同；然而在基于磁盘的表上，主键比其他索引的速度快很多。

实际上，除了内存表的速度通常特别快之外，内存表与基于磁盘的表没什么区别。

非持久内存表

当服务器关闭时，不会将非持久内存表写入磁盘。因此，无论服务器在任何时候正常关闭还是异常关闭，非持久表中的数据都会丢失。不会对它们包含的数据进行记录或执行检查点操作。这使得它们不可恢复，但是比持久表的运行速度明显快得多。

非持久表主要是作为“暂存区”表时很有用。例如，您可以复制持久表中的数据，运行一系列分析，然后废弃非持久副本。您也可以执行一系列变换，然后将最终结果复制回持久表。

有两种不同类型的非持久内存表：瞬态表和临时表。在第 13 页的2 章，『临时表和瞬态表』中详细讨论了这两种非持久内存表之间的差别。在本节中我们将只提供简要概述。

瞬态表

瞬态表将一直保持到数据库服务器关闭为止。多个用户可以使用同一个瞬态表，并且每个用户可以查看所有其他用户的数据。

与持久内存表相比，瞬态表具有一些局限性。例如，将外键（引用约束）与瞬态表配合使用时就具有一些局限性。此外，不会将瞬态表复制到 HotStandby 辅助服务器中。

临时表

临时表中的数据仅对于插入此数据的连接可视，并且仅在连接持续时间内才会保留此数据。与专用暂存区一样，其他用户不能查看临时表。

除了不用记录以外，临时表也不使用任何类型的并行控制机制（例如，锁定记录），这使得它们甚至比瞬态表的速度更快。

内存表能够提高性能的原因

如果数据存储在基于磁盘的表中，那么必须将此数据读取到内存中之后才能使用，并且使用之后必须将它写回磁盘。使用内存表可以提供更高的性能，这是因为所有数据始终都驻留在主内存中，因此服务器可以使用效率更高的技术使访问和处理数据时获得最佳性能。

当然，几乎对于任何数据库服务器来说，如果它具有更多内存并且可以将它的更多数据存储在服务器的高速缓存中，那么它的执行速度将更快。但是，solidDB 主内存引擎的高性能内存技术不仅仅只是将数据复制到内存中。solidDB 主内存引擎还使用经过优化的索引结构来处理完全存储在内存中的数据。solidDB 主内存引擎还考虑了内存表产生的问题，例如，当表增大或缩小时内存将“分段”。

决定将哪些表指定为内存表的方法

理想情况下，您的计算机具有足够的内存，可以将所有表都存储在内存中，从而为数据库事务提供可以达到的最佳性能。然而在现实生活中，大多数用户不得不选择将一部分表存储在内存中，而将其余表存储在磁盘中。

当然，如果您不能将所有表都放入内存中，那么您可能希望尽量将最常用的数据放入内存中。很显然，应该将频繁使用的小型表放入内存中，而将很少使用的大型表保存在磁盘上。但是，对于其他可能存在的组合情况如何呢？例如，对于大量使用的大型表或者不是大量使用的小型表如何呢？

对于此问题，最好是考虑对表的访问“密度”。一个表的每秒每兆字节的访问数越高，那么您最好是将此表放入内存中。有关更详细的算法，请参阅第 19 页的『用于选择要存储在内存中的表的算法』。

一旦您决定了将一个表存储在内存中，那么必须选择是将数据存储在持久表、瞬态表还是临时表中。下面说明了一些基本规则。请注意，这些规则只是一些准则而不是硬性规则。在做出最终决策之前，您应当阅读对于瞬态表和临时表的详细描述。

请注意，我们所说的“服务器会话”是指单次“运行”服务器。即，从此服务器启动开始，直到它被有意关闭或者由于意外原因（例如，电源故障）而停机为止。一个“连接”的生命周期是从单个用户连接至服务器开始，直到此用户断开此连接为止。（一个用户可以建立多个连接，但是每个连接都是独立的。）

可以通过以下方法来决定最适合于您使用的表类型：请您依次回答下列问题，直到您对所提出的问题首次回答“是”为止。

1. 您需要使数据在服务器下一次启动时再次可用吗？如果需要，那么请使用持久表。

2. 您需要将数据复制到辅助 HotStandby 服务器吗？如果需要，那么请使用持久表。

3.

您仅在当前服务器会话期间需要数据，但是数据必须可供多个用户（或者同一个用户的多个连接）使用吗？如果是这种情况，那么请使用瞬态表。

4.

如果上述规则都不适用，那么请使用临时表。

此外，在做出最终决策之前，您应当阅读有关临时表和瞬态表的详细信息。临时表和瞬态表具有一些限制（例如，对于外键（引用约束）的限制），它们可能会影响您的决策。

用于指定一个表存储在内存中的方法

可以通过两种方法来显式指定表是位于内存中还是磁盘上。缺省情况下，系统将创建基于磁盘的表。

1. 使用 CREATE TABLE 或 ALTER TABLE 命令的 STORE 子句。例如:

   ```
   CREATE TABLE employees (name CHAR(20)) STORE MEMORY;
   CREATE TABLE ... STORE DISK;
   ALTER TABLE network_addresses SET STORE MEMORY;
   ```

   （有关 CREATE TABLE 和 ALTER TABLE 语句语法的更多信息，请参阅《IBM solidDB SQL 指南》。）
2. 在 solid.ini 文件中指定缺省值。为了方便起见，如果您创建的大多数新表或所有新表都应该是同一类型（例如，内存表），那么您可以让服务器自动使用此存储器类型（除非另有声明）。要指定新表的缺省类型，请在 solid.ini 配置文件中设置以下参数:

   ```
   [General]
   DefaultStoreIsMemory=yes
   ```

   如果此参数设置为“yes”，那么会将新表作为内存表来创建（除非在 CREATE TABLE 语句中另有声明）。如果此参数设置为“no”，那么会将新表作为基于磁盘的表来创建（除非在 CREATE TABLE 语句中另有声明）。与 solid.ini 文件中的其他参数一样，如果更改了此参数，那么新值将在下一次启动服务器之后才会生效。有关 **DefaultStoreIsMemory** 参数的更多信息，请参阅《IBM solidDB 管理员指南》。

注:

这些指示信息仅适用于持久表。即使您不使用 STORE MEMORY 子句，也会将声明为临时表或瞬态表的那些表自动存储在内存中。

内存消耗量

了解和控制内存消耗量是很重要的，这是因为如果内存数据库或者服务器进程用尽了系统中的所有可用虚拟内存，您就无法添加或更新数据。如果服务器用尽了所有物理内存并且开始使用虚拟内存，那么服务器虽然将继续运行，但是性能将显著下降。

内存数据库主内存使用量与标准 solidDB 不同。内存数据库驻留在它自己的内存池中。有关 solidDB 内存消耗量的更多信息，请参阅《IBM solidDB 管理员指南》。

solidDB 主内存引擎提供了一些命令和配置参数来帮助您监视和控制内存数据库和服务器进程的内存消耗量。这些命令和参数主要用于服务器的内存数据库功能部件而不是整个服务器。

监控内存消耗量

有多个 ADMIN COMMAND 可用来监控内存消耗量。简单来说，这些命令是:

- ADMIN COMMAND 'info imdbsize';
- ADMIN COMMAND 'info processsize';
- ADMIN COMMAND 'pmon mme';
- ADMIN COMMAND 'memory';

下面说明了这些命令。

命令

ADMIN COMMAND 'info imdbsize';

将返回当前已分配给内存数据库表和索引使用的内存量。返回的值属于 VARCHAR 类型，它表示服务器使用的内存量（按千字节计）。请注意，此命令返回的是所使用的虚拟内存量，而不是返回所使用的物理内存量。

随着时间推移，imdbsize 会增大，这是因为只能在分配单元中将数据返回给操作系统，而分配单元需要在完全不再使用之后才能将它们返回给操作系统。

ADMIN COMMAND 'info imdbsize'; 报告中不包含瞬态内存分配情况（例如，SQL 执行情况图）。

命令

ADMIN COMMAND 'info processsize';

将返回虚拟内存进程大小，即，内存数据库进程使用的数据库服务器的完整地址空间大小。返回的值属于 VARCHAR 类型，它表示此进程使用的内存量（按千字节计）。请注意，此命令返回的是所使用的虚拟内存量，而不是返回所使用的物理内存量。

还提供了多个性能计数器，它们包含与内存数据库服务器相关的运行时信息。通过输入

ADMIN COMMAND 'pmon mme';

命令将生成这些计数器的当前值的以下列表。

```
RC TEXT
-- ----
 0 Performance statistics:
```

```
 0 Time (sec)                          30    21    Total
 0 MME current number of locks       :     0     0        0
 0 MME maximum number of locks       :     0     0        0
 0 MME current number of lock chains :     0     0        0
 0 MME maximum number of lock chains :     0     0        0
 0 MME longest lock chain path       :     0     0        0
 0 MME memory used by tuples         :     0     0        0
 0 MME memory used by indexes        :     0     0        0
 0 MME memory used by page structures:     0     0        0
10 rows fetched.
```

在性能统计信息列表中，由元组、索引和页面结构使用的内存量是按千字节给定的。

命令

ADMIN COMMAND 'memory';

将只报告动态分配的堆内存量。按堆进行内存分配时，将从未使用的内存区（称为堆）的一个大型内存池中分配内存。可以在运行时确定堆内存分配大小。ADMIN COMMAND 'mem'; 报告中包含了瞬态内存分配情况（例如，SQL 执行情况图）。

控制内存消耗量

有三个配置参数 ImdbMemoryLimit、ImdbLowPercentage 和 ImdbMemoryWarningPercentage 可用来控制内存数据库的内存消耗量。这些参数位于 solid.ini 文件的 [MME] 节中。

此外，还有 ProcessMemoryLimit、ProcessMemoryLowPercentage、ProcessMemoryWarningPercentage 和 ProcessMemoryCheckInterval 这四个配置参数用来控制进程的内存消耗量。这些参数位于 solid.ini 文件的 [SRV] 节中。

如果违反了 IMDB 和进程限制，那么这些违例将记录在 solmsg.out 日志文件中。

如上所示，有两个内存大小限制，即 ImdbMemoryLimit 和 ProcessMemoryLimit。每当超过内存限制时，就会记录一个系统事件。《IBM solidDB SQL 指南》中描述了这些系统事件。

ImdbMemoryLimit

ImdbMemoryLimit 参数指定可以分配给内存表（包括临时表、瞬态表和“正常的”内存表）以及关于这些内存表的索引的最大虚拟内存量。

ImdbMemoryLimit 参数的缺省值为 0，它表示“无限制”。我们强烈建议您不要使用缺省值。应将此参数设置为一个能确保将内存数据全部放入物理内存中的值。当然，您必须考虑下列因素:

- 计算机中的物理内存量
- 操作系统使用的内存量
- solidDB 程序本身使用的内存量
-

为 solidDB 服务器的高速缓存预留的内存量（CacheSize solid.ini 配置参数）

•

同时在服务器中运行的连接、事务和语句所需要的内存量。服务器中的并发连接和活动语句越多，服务器需要的工作内存就越多。通常，应当为服务器中的每个客户机连接至少分配 0.5MB 内存。

•

正在计算机中运行的其他进程（程序和数据）使用的内存

达到限制时（即，内存表使用了由 ImdbMemoryLimit 指定的所有内存），服务器就会禁止对内存表执行更新（UPDATE）操作。（在达到限制之前，服务器将禁止创建新的内存表和对这些表执行插入（INSERT）操作。有关更多详细信息，请参阅对 ImdbLowPercentage 参数的描述。）

示例:

```
[MME]
ImdbMemoryLimit=1000MB
```

ImdbLowPercentage

ImdbLowPercentage 变量将设置一个“低水位标记”，用内存所占的百分比（即，ImdbMemoryLimit 的百分比）来表示。当服务器消耗的内存量达到所指定的百分比时，服务器就会开始限制活动，以防止内存消耗量继续增加。例如，如果 ImdbMemoryLimit 为 1000 MB，并且 ImdbLowPercentage 为 90%，那么当分配给内存表的内存量超过 900 MB 时，服务器就会开始限制活动。尤其是，服务器将执行下列操作:

•

禁止进一步创建内存表（包括临时表和瞬态表）以及关于内存表的索引。

•

禁止插入（INSERT）内存表。

达到上限本身（即，ImdbMemoryLimit）时，服务器还将禁止对内存表中的记录执行更新（UPDATE）操作。

ImdbLowPercentage 的有效值范围是 60 到 99（按百分比计）。

ImdbMemoryWarningPercentage

ImdbMemoryWarningPercentage 参数为 IMDB 内存大小设置一个警告限制。此警告限制用 ImdbMemoryLimit 参数值的百分比来表示。超过 ImdbMemoryWarningPercentage 限制时，就会产生一个系统事件。

达到 ImdbMemoryLimit 之后应执行的操作

如果产生了一条错误消息指出已经达到了此限制，那么您需要立即执行强有力的操作。您必须解决紧迫的问题和长期的问题。紧迫的问题是防止用户遇到严重错误以及在关闭服务器之前释放一些内存，以便当您重新启动服务器时不会遇到相同的情况（内存不足）。长期问题是要确保将来扩展表时不会再次遇到这种情况。

要解决紧迫的问题，通常应执行下列操作:

1.

通知用户应当与服务器断开连接。这样做将完成以下两项任务：1) 将使得在情况恶化时受影响的人数最少；2) 如果断开连接的任何用户先前在使用临时表，那么断开连接之后就会释放内存。您可能希望通过一项策略或错误检查代码来确保，用户和/或程序在发现此错误时将尝试适时地断开连接。

2.

如果没有足够多的临时表使得内存消耗量有太大差别，那么请删除某些瞬态表索引或者废弃瞬态表本身（如果它们存在）。

如果没有足够多的临时表和瞬态表使得内存消耗量有太大差别，那么您将需要执行更彻底的操作。

1.

删除内存表的一个或多个索引。

2.

关闭服务器。

3.

如果内存中完全没有需要废弃的对象（例如，内存中只有“正常的”内存表，这些表都没有索引，并且这些表都具有有价值的数据），那么请在重新启动服务器之前稍微增大 ImdbMemoryLimit。这可能会强制服务器开始将虚拟内存分页，从而显著降低性能，但是它将允许您继续使用服务器并解决长期问题。如果您先前将 ImdbMemoryLimit 设置为稍微低于最大值，那么您现在可以将它稍微提高，而不强制系统开始将虚拟内存分页。

4.

重新启动服务器。

5.

在您有时间解决长期问题之前，请将使用此系统的人数降到最低限度。请确保用户不创建临时表或瞬态表，直到解决了长期问题为止。

如果您已经解决了紧迫问题并且已确保服务器至少有一点可用内存，那么您就可以开始解决长期问题。

要解决长期问题，您需要减少存储在内存表中的数据量。这可以通过减小内存表数目或大小（包括临时表和瞬态表在内）或者减小内存表的索引数目来实现。

如果问题只是由于大量使用了临时表（或瞬态表）而导致的，那么您可能只需确保没有太多会话同时创建太多的大型临时表（或瞬态表）。

如果问题是由于“正常的”内存表使用了太多内存，而您又不能增大可用于服务器的内存量而导致的，那么您必须将一个或多个表从主内存移到磁盘中。值得庆幸的是，这不是一项很难的任务。要将一个表从内存移到磁盘中，请执行下列操作：

1.

创建一个基于磁盘的空表，使其结构与内存中的一个表的结构相同，但是名称不同。

2.

将此内存表中的信息复制到基于磁盘的表中。[1]

3. 废弃内存表。

4. 重命名基于磁盘的表，使它具有现在已废弃的内存表的原始名称。

有意将 ImdbMemoryLimit 设置为稍微低于您实际上具有的最大内存，这是一项很好的预防措施。这样，如果您用尽了内存并且没有不需要且可除去的内存表或索引，那么可以稍微增大 ImdbMemoryLimit，然后在具有足以满足长期需要的可用内存的情况下重新启动服务器。

还应记住，可以使用 ImdbMemoryWarningPercentage 参数来提醒您增大内存消耗量。

并不是所有情况下都要求您减少内存表的数目。在某些情况下，最可行的解决方案可能是只需在计算机中增加更多内存而已。

还应记住，防止发生问题可能比解决问题更好。强烈建议您将 ImdbMemoryLowPercentage 参数设置为一个适当的值，以便在您用尽所有可用于内存表的内存之前为您提供一个值得信赖的警告。

ProcessMemoryLimit

通过 ProcessMemoryCheckInterval 参数来启用和禁用进程内存限制。如果 ProcessMemoryCheckInterval 参数值为 0，那么将不存在进程内存限制。如果需要控制服务器进程的虚拟内存总大小，那么可以在定义 ProcessMemoryCheckInterval 参数之后使用此限制。

ProcessMemoryLimit 参数指定可以分配给内存数据库进程的最大虚拟内存量。ProcessMemoryLimit 的出厂值为 1G（即，1 千兆字节）。请将此参数设置为一个能确保将内存数据库进程全部放入物理内存中的值。当然，您必须考虑下列因素:

- 计算机中的物理内存量
- 操作系统使用的内存量
- 内存表（包括临时表、瞬态表和“正常的”内存表）以及关于这些内存表的索引所使用的内存量。
-

---

1. 如果您尝试使用单个 SQL 语句（INSERT INTO ...VALUES SELECT FROM）将一个大型表的记录复制到另一个表中，那么请记住，整个操作是在一个事务中执行的。仅当所有数据都可以放入服务器的高速缓存中时，上述这样一个操作才会生效。如果事务大小的增长速度比高速缓存大小的增长速度更快，那么性能将显著下降。因此，强烈建议您在较小的事务（例如，每个事务只有几千行）中使用简单的存储过程或应用程序将一个大型表的数据复制到另一个表中。

   还应注意，中间表不需要索引。在成功复制数据之后，应在新的“实际表”中重新创建索引。

为 solidDB 服务器的高速缓存预留的内存量（CacheSize solid.ini 配置参数）

- 同时在服务器中运行的连接、事务和语句所需要的内存量。服务器中的并发连接和活动语句越多，服务器需要的工作内存就越多。通常，应当为服务器中的每个客户机连接至少分配 0.5MB 内存。
- 正在计算机中运行的其他进程（程序和数据）使用的内存

达到限制时（即，内存数据库进程使用了由 ProcessMemoryLimit 指定的所有内存），服务器将只接受管理命令（ADMIN COMMAND）。可以使用 ProcessMemoryWarningPercentage 参数来提醒您增大内存消耗量。

ProcessMemoryLowPercentage

此参数对进程总大小设置一个限制。此限制是用 ProcessMemoryLimit 参数值的百分比表示的。在超过此限制之前，您已经超过了使用 ProcessMemoryWarningPercentage 参数定义的警告限制，并且接收到一条警告消息。超过 ProcessMemoryLowPercentage 限制时，就会产生一个系统事件。

ProcessMemoryWarningPercentage

此参数对进程总大小设置一个警告限制。此警告限制是用 ProcessMemoryLimit 参数值的百分比表示的。超过 ProcessMemoryWarningPercentage 限制时，就会产生一个系统事件。

ProcessMemoryCheckInterval

将定期检查进程大小限制。可使用 ProcessMemoryCheckInterval 参数来设置检查时间间隔。时间间隔是按毫秒指定的。

如果值不为零，那么其最小值应为 1000（毫秒）。只允许值为 0 或者大于等于 1000（1 秒）。如果给定的值大于 0 但小于 1000，就会产生一条错误消息。

出厂值为 0，这就意味着将禁止检查进程大小。

计算磁盘空间需求

当计算存储您的数据库所需要的磁盘空间量时，必须考虑存储内存表以及基于磁盘的表所需要的空间量，这是因为当服务器关闭时，它会将内存数据存储到磁盘驱动器中。（当服务器重新启动时，它会将此信息重新读取到内存中。）

还应注意，因为服务器在关闭时必须将内存数据写入磁盘驱动器，因此关闭具有内存表的服务器比只具有基于磁盘的表的服务器需要更长时间。同样，当服务器启动时必须将数据从磁盘驱动器重新装入内存中，因此，如果服务器具有内存表，那么通常需要更长时间来启动。

是否符合标准

内存表功能部件并不是 SQL-99 的 ANSI 标准的一部分。

内存表的局限性

物理内存和虚拟内存

最明显的局限性是内存数据库表的总大小不能超过可用虚拟内存量。

要点:

由于虚拟内存将频繁地与磁盘进行交换，因此使用虚拟内存会削弱内存表的一部分优势。强烈建议您限制内存表，使它小于可用物理内存的大小（而不是可用虚拟内存的大小）。

在计算表所需要的空间量时，请不要忘记应考虑“BLOB”数据。通常，应将 BLOB 数据保存在基于磁盘的表中，因为 BLOB 列的最大大小在主内存表上将显著减小。

请记住，存储一个表所需要的空间量不仅包括存储表中的数据需要的空间，而且包括此表的任何索引（其中包括为了支持主键和外键约束而创建的任何索引）所需要的空间。此外，表在内存中比在磁盘上将占用更多空间。

如果服务器在尝试分配内存时（例如，在执行 INSERT 或 ALTER TABLE 操作期间扩展表）用尽了虚拟内存，那么将产生一条错误消息。

将表从内存表更改为基于磁盘的表或者从基于磁盘的表更改为内存表

如果一个表是空的，那么可以将此表的类型从内存表改变为基于磁盘的表，或者从基于磁盘的表改变为内存表。可使用

```
ALTER TABLE table_name SET STORE MEMORY | DISK
```

命令来完成此任务。如果此表中包含数据，那么您需要创建一个具有其他名称的新表，用于将数据复制到此表中。将数据复制到此新表之后，可以废弃旧表并使用

ALTER TABLE *current_table_name* SET TABLE NAME *new_table_name*

命令将此新表重命名。

事务隔离

“可序列化”隔离级别不受支持。

不能在事务隔离级别为“可序列化”的事务中使用内存表。内存表支持的事务隔离级别为“可重复读”和“落实读”。

在 HotStandby 辅助服务器上，事务隔离级别始终为“落实读”。

如果您要使用 HotStandby 并且已连接至 HotStandby 辅助服务器，那么当您从内存表中读取数据时，会自动将事务隔离级别设置为“落实读”，即使您先前已指定了“可重复读”也是如此。（在主服务器或辅助服务器上，内存表都不支持“可序列化”隔离级别。）

其他内存引擎增强功能

除了内存表以外，solidDB 主内存引擎还有两项可提高性能的特征。第一，读操作不用等待磁盘访问，即使系统在执行检查点操作和记录事务等活动时也是如此。第二，它将进行宽松的事务记录，因此写操作也不会等待磁盘访问。有关宽松的事务记录与严格的事务记录的讨论，请参阅《IBM solidDB 管理员指南》。

将 solidDB 链接库访问和 HotStandby 与 solidDB 内存引擎配合使用

链接库访问

链接库访问以一个可链接库的形式提供服务器。用户可以将应用程序直接链接至此库并通过执行函数调用来对它进行访问，而不需要使用网络通信协议。

链接库访问与 solidDB 主内存引擎的内存表功能部件兼容。可以在链接库访问服务器中创建内存表。

有关链接库访问的更多信息，请参阅《IBM solidDB 链接库访问用户指南》。

HotStandby

solidDB 高可用性组件提供了"热备用"能力。这意味着您的数据库服务器与另一个服务器是成对的，它们的数据自动保持同步。如果其中一台服务器发生故障，那么可以继续使用另一台服务器。

solidDB 主内存引擎的内存功能部件与 solidDB HotStandby 兼容。

持久内存表（即，未特地指定为临时表或瞬态表的内存表）将从 HotStandby 主服务器复制到辅助服务器。

请注意，临时表和瞬态表不会复制到辅助服务器中。

与先前的 solidDB 产品的不兼容性

solidDB 主内存引擎几乎与先前版本的 solidDB 产品完全兼容。但是，也有一些例外。有关兼容性问题的信息，请参阅 releasenotes.txt 文件。

2 临时表和瞬态表

临时表和瞬态表比标准的内存表提供了更高的性能。但是，并不会持久保存临时表和瞬态表中的数据。因此，如果您想持久存储临时表或瞬态表中的数据，那么可能需要将此数据复制到另一个表中。

缺省情况下，当您创建内存表时，此表是一个持久表。当服务器关闭时，此表将被写入磁盘中；当服务器再次启动时，又将从磁盘中读取此表。但是，solidDB 主内存引擎提供了以下两种类型的非持久内存表：临时表和瞬态表。这两种类型的表都打算用于存放"临时"数据。

由于下列原因，临时表和瞬态表都提供了较高的性能:

- 临时表和瞬态表中的数据仅存储在内存中，决不会写入磁盘中。（如果您关闭服务器然后将它重新启动，或者服务器异常终止，那么临时表或瞬态表中的数据将丢失。对于临时表，在用户会话结束时就会废弃此表中的数据 – 它更加不会保留到服务器关闭为止。）
- 临时表和瞬态表不会将事务数据记录到磁盘中。（在服务器异常终止之后，不可恢复这些表中的数据。）
- 当服务器执行定期"检查点"操作时（此操作会将数据库数据写入磁盘驱动器），不会将临时表和瞬态表中的数据写入磁盘。（有关检查点的更详细说明，请参阅《IBM solidDB 管理员指南》。）
- 临时表和瞬态表不仅使用 solidDB 为内存表提供的高性能技术，而且比常规内存表使用效率更高的数据存储结构。
- 临时表具有更大的性能优势。会话（连接）不会互相查看对方在临时表中的记录，因此临时表不需要进行复杂的并行控制（例如，不需要检查是否存在对于此表中的记录的锁定冲突）。

将临时表和瞬态表作为"暂存区"特别有用。例如，您可以复制持久表中的数据，当此数据复制到临时表中之后对它执行一系列集中操作，然后将结果存储回持久表中。这样您就可以使性能达到最佳，同时在您完成任务时还保留一部分或所有数据。即使您的工作因某种原因而中断，原始数据仍然会安全地保存在此持久表中，您可以重新开始处理此数据。

临时表与瞬态表之间的主要差别是，临时表中的数据仅对于单个连接可视，而瞬态表中的数据对于所有用户都可视。

下面是有关这两种类型的表中每种表的更具体信息。临时表和瞬态表有许多共同点，因此，在下面两节中您将发现一些重复内容。我们单独而且充分描述了每种类型的表，然后明确指出了它们之间的差别。在第 20 页的『临时表和瞬态表的性能调整信息』中进一步讨论了它们之间的差别。

临时表

临时表中的数据的可视性非常有限，并且持续时间非常短。

有限的可视性

数据具有有限的可视性，这是因为只有插入此数据的会话（连接）才能查看此数据。如果您的会话创建一个临时表并且将数据插入此表中，那么即使您为其他用户会话授予对此表的特权，这些会话也不能查看您的数据。注意，多个会话可以同时使用同一个表，但是每个会话将只查看它自己的数据。（请注意，由于每个会话将只查看它自己的数据，因此您不需要与其他会话进行协调就可以确保将唯一值插入表中，即使此表具有唯一约束也是如此。例如，如果您创建一个对"标识"列具有唯一约束的临时表，那么您和另一个会话都可以插入将标识设置为 1 的记录。由于每个会话只能查看它自己的数据，因此，执行诸如 UPDATE 和 DELETE 等操作将只影响会话自己的数据。

有限的持续时间

数据都具有有限的持续时间，因为一旦您退出当前会话（即，与服务器断开连接），就会废弃数据。如果您再次连接，将看不到您的数据。

需要明白的是，"临时表"中的"临时"是指数据是临时的，而不是指表本身。实际上，服务器是将临时表的定义而不是数据存储在服务器的系统表中，即使在您与服务器断开连接之后仍将保留此定义。因此，如果您稍后重新连接至服务器，您会发现此表仍然存在，但它是空的。因此，一旦您创建了临时表，在将来的会话中就不需要再次创建此表。实际上，如果您或者其他用户试图创建一个与现有临时表同名的临时表，那么将产生一条错误消息。如果您认为一旦您断开连接，"临时表"（而不仅仅是其中的数据）就会消失，那么此行为可能会让您感到意外。

当然，由于这些表持久存在（尽管其中的数据并不持久存在），因此在您不再需要这些表之后，应使用 DROP TABLE 命令来废弃表定义。

因为表持久存在，所以如果您导出数据库模式定义，那么输出中将包含用于重新创建临时表的命令。

因为当用户断开连接时就会清除会话的临时表，所以在一个具有许多临时表数据的会话断开连接之后的一段时间内，服务器的 CPU 使用量可能会较高。

附加局限性

下面是临时表存在的一些附加局限性。

- 当您使用 HotStandby 组件时，不会将临时表中的数据复制到辅助服务器中。但是请注意，临时表定义本身是会被复制到 HotStandby 辅助服务器中的。因此，当您需要故障转移到辅助服务器时，不需要重新创建先前已创建的任何临时表，但是需要重新创建这些临时表中的所有数据。
- 在高级复制系统中，无法将临时表用作"主控"表，但是可以将它们用作副本表。
- 临时表对于可以如何将它们与引用约束配合使用具有一些限制。一个临时表可以引用另一个临时表，但是不能引用任何其他类型的表（即，瞬态表或持久表）。除了临时表之外，其他类型的表不能引用临时表。请参阅本章中的"引用约束"表。

除了本节中所列示的局限性之外，临时表与正常的内存表（即，持久内存表）的行为相似。例如，

- 临时表可以具有它们自己的索引。
- 可以在视图中使用临时表。
- 临时表可以具有用于它们的触发器。

- 临时表可以包含 BLOB 列，但是这些列的长度不能超过几千字节。
- 临时表位于特定目录中并且采用特定模式。
- 需要具有特权才能使用临时表；换句话说，临时表的创建者可以授予和撤销对于此表的特权。DBA 也可以授予和撤销对此表的特权。但是请记住，当一个会话将数据存入临时表时，任何其他会话都看不到此临时表中的数据，即使后面这个会话是由 DBA 或者已被授予了对此临时表的 SELECT 特权的用户发出的也是如此。因此，授予对一个表的特权时，只是为其他用户授予了使用此表的权限，但他们不能使用此表中的数据。请注意，对于临时表的缺省特权与对于持久表的缺省特权相同。

要创建临时表，请使用下面所显示的语法，其中“<...>”表示对于任何其他类型的表都相同的语法。

```
CREATE [GLOBAL] TEMPORARY TABLE <...>;
```

有关 CREATE TABLE 命令的完整语法，请参阅《IBM solidDB SQL 指南》。

临时表始终是一个内存表。如果您使用 STORE DISK 子句，那么服务器将报告错误。如果您使用 STORE MEMORY 子句或者完全省略 STORE 子句，那么服务器会将临时表作为内存表来创建。

solidDB 的临时表实现完全符合“全局临时表”的 ANSI SQL:1999 标准。无论是否指定了 GLOBAL 关键字，solidDB 的所有临时表都是全局表。solidDB 不支持按照 ANSI 定义的“本地临时表”。

瞬态表

瞬态表中的数据的持续时间有限。服务器只将此数据保存到服务器关闭为止。

瞬态表中的数据与持久表中的数据具有相同的“作用域”或可视性。如果其他用户具有适当的特权。那么这些用户的会话可以查看插入到瞬态表中的数据。

瞬态表具有一些局限性:

- 当您使用 HotStandby 组件时，不会将瞬态表中的数据复制到辅助服务器中。请注意，瞬态表本身（而不是它们的数据）将被复制到 HotStandby 辅助服务器中。因此，当您需要故障转移到辅助服务器时，不需要重新创建先前已创建的任何瞬态表，但是需要重新创建这些瞬态表中的所有数据。
- 瞬态表对于可以如何将它们与引用约束配合使用具有一些限制。瞬态表可以引用其他瞬态表，还可以引用持久表。但是，它们不能引用临时表。临时表和持久表都不能引用瞬态表。请参阅本章中的“引用约束”表。
- 在高级复制系统中，无法将瞬态表用作“主控”表，但是可以将它们用作副本表。

除了本节中所列示的局限性之外，瞬态表与“正常的”内存表（即，持久内存表）的行为相似。例如:

-

可以在视图中使用瞬态表。

- 瞬态表可以具有它们自己的索引。
- 瞬态表可以具有用于它们的触发器。
- 瞬态表可以包含 BLOB 列，但是这些列的长度不能超过几千字节，与所有内存表的情况相同。
- 适用于瞬态表的特权。
- 瞬态表位于特定目录中并且采用特定模式。

如果您导出一个具有瞬态表的数据库，那么将导出瞬态表中的数据（以及表的结构）。

要创建瞬态表，请使用下面所显示的语法，其中“<...>”表示对于任何其他类型的表都相同的语法。

```
CREATE TRANSIENT TABLE <...>;
```

（有关 CREATE TABLE 命令的完整语法，请参阅《solidDB SQL 指南》。）

瞬态表始终是内存表。如果您使用 STORE DISK 子句，那么服务器将报告错误。如果您使用 STORE MEMORY 子句或者完全省略 STORE 子句，那么服务器会将瞬态表作为内存表来创建。

请注意，服务器实际上是将瞬态表的定义而不是数据存储在服务器的系统表中，即使在服务器关闭之后仍将保留此定义。如果您稍后重新启动服务器，您会发现此表仍然存在，但是其中的数据已不存在。因此，一旦您创建了瞬态表，以后就不需要再次创建此表。实际上，如果您或者其他用户试图创建一个与现有瞬态表同名的瞬态表，那么将产生一条错误消息，即使自从最初创建具有此名称的表以来已经关闭然后重新启动了服务器也是如此。如果您认为一旦您关闭服务器，“瞬态表”就会消失，那么此行为可能会让您感到意外。

当然，由于瞬态表持久存在（尽管其中的数据并不持久存在），因此在您不再需要此表之后，应使用 DROP TABLE 命令来废弃此表。

可以使用 solload 实用程序将数据导入瞬态表中。

瞬态表并不是按照 SQL 的 ANSI 标准进行定义的。瞬态表是 solidDB 对 SQL 标准的扩展。

临时表与瞬态表之间的差别

临时表与瞬态表之间的主要差别如下:

-

瞬态表允许系统中的所有会话（连接）查看同一个数据，而临时表只允许创建了数据的用户查看该数据。

由于用户可以访问同一个数据，因此瞬态表将使用并行控制。目前，瞬态表仅支持悲观并行控制（“锁定”）。

由于临时表不使用并行控制，因此临时表比瞬态表的运行速度更快。

•

瞬态表中的数据将一直保存到服务器关闭为止，而临时表中的数据仅保存到用户注销会话为止。这就意味着，如果一个会话将数据插入瞬态表中，那么其他会话也可以查看此数据，即使在此数据的创建者断开连接之后仍然可以查看。

•

可使用 solexp 工具来导出瞬态表中的数据，而临时表中的数据则不行。

•

这两种类型的表的引用完整性规则不同。

下表说明了允许哪些表类型引用其他类型。例如，如果允许瞬态表具有引用持久表的外键，那么“瞬态子代”行与“持久父代”列交叉处的单元格中的值将为“是”。如果不允许存在外键约束，那么您在此单元格中将看到一条短划线（“-”）。

表 3. 引用约束

被引用的表 **引用表**	**基于磁盘的持久表**	**持久内存表**	**瞬态表**	**临时表**
持久 基于磁盘的表	是	是	-	-
持久 内存表	是	是	-	-
瞬态 表	是	是	是	-
临时 表	-	-	-	是

每种类型的表都可以引用它本身。此外，瞬态表可以引用持久表，但是持久表不能引用瞬态表。所有其他组合情况都无效。

3 优化和调整服务器

本章提供了有关如何使用 solidDB 主内存引擎中提供的功能部件来优化和调整服务器。

用于选择要存储在内存中的表的算法

如果您有足够的内存可用于将某些表（但不是所有表）放入内存中，那么以下策略将指导您选择要放入内存中的表。

总的原则是应当考虑访问的“密度”。很显然，访问频率越高，访问“密度”就越大。同样，表越大，用给定的每秒访问数表示的访问“密度”就越低。

访问密度是按照每秒每兆字节访问单元数进行度量的，我们通常将它书写为“行数/MB/秒”。为了简单起见，我们假定每行存在一个访问。例如，假定有一个大小为 1 MB 的表，您在 10 秒钟之内访问了 300 行，那么访问密度为:

300 行/1 MB/10 秒 = 30 行/MB/秒

再举一个例子，假定您有一个大小为 500KB 的表，您每秒访问 300 行，那么访问密度为:

300 行/0.5 MB/秒 = 600 行/MB/秒

很明显，第二个表比第一个表的访问密度更高，如果您只能将其中一个表放入内存中，那么应该将第二个表放入内存中。

此公式稍微有点简单。

1.

   您可能希望考虑每次访问的字节数。这通常是指平均行大小。但是，如果您使用二进制大对象，或者如果服务器通过只读取索引而不需要读取整个表就可以找到它需要的所有信息，那么平均行大小可能也不同。

   请注意，因为服务器通常是按“块”（一个块的大小通常为 8KB）的倍数从磁盘中读取数据，所以“每次访问的字节数”或者“每行的字节数”提供的数字只是比没有考虑它们的公式稍微准确一些。无论您读取 10 个字节一行还是 2000 个字节一行，服务器完成的工作量大致相同。

2.

   请记住，考虑表的大小时，还必须考虑该表所有索引的大小。每当您添加索引时，就会添加所存储的有关此表的更多数据。而且，当您对一个表添加外键约束时，如果不存在索引，那么服务器将创建一个适当的索引以提高对此表执行某些类型的查询操作的速度。当您计算一个表在内存中占用空间的大小时，必须同时考虑此表及其所有索引和所有 BLOB。

在计算了所有表的访问密度之后，将这些表按访问密度从最高到最低的顺序排列。然后，从访问密度最高的那个表开始，将列表中的表依次指定为内存表，直到用尽所有可用物理内存为止。

然而，此过程并不是这么简单，因为上述描述的前提条件是您具有准确的信息，并且您随时可以将基于磁盘的表更改为内存表（反之亦然）。但实际上您可能并不知道计算机上的可用内存总量。您可能意外指定了过多的内存表，超出了物理内存中计算机用于存放内存表的空间。因此，这些表可能会被换存到磁盘中。这可能会显著降低性能。此外，您可能实际上并不知道每个表的访问频率，直到此表中包含大量数据为止。然而，此 solidDB 服务器的当前版本要求您在创建表时就将此表指定为内存表或者基于磁盘的表，然后才向它添加任何数据。因此，必须根据您对每个表的使用量、每个表的大小以及可用内存量的估计来进行计算。同时它还假定平均访问密度不会随着时间推移而更改。

此方法还假定您将来也不打算添加更多表，并且表的大小也不会增大。在典型情况下，您不应当用尽所有内存 － 您应当保留足够的空间以考虑到表的大小可能会增大，并且还应保留一点误差，以便不用尽内存。

尽管如此，此算法还是对您选择要要放入内存中的表提供了基本指导。

要点:

由于虚拟内存可能会频繁地与磁盘进行交换，因此使用虚拟内存会削弱内存表的优势。始终应确保整个 DBMS 进程符合计算机的物理内存情况。

临时表和瞬态表的性能调整信息

注意:

在决定是使用临时表还是瞬态表时，请记住，临时表和瞬态表中的数据都是临时数据而不是持久数据。不会将这些数据存储到磁盘中。如果您要使用 HotStandby，那么不会将这些数据复制到 HotStandby 辅助服务器中。也不会将这些数据写入事务日志，因此在服务器异常终止时无法恢复这些数据。如果您不容许丢失这些数据并重新开始工作，那么请不要使用临时表或瞬态表。

如果您要决定是使用临时表还是瞬态表，请记住下列主要事项:

- 临时表中的数据不会持续保存到会话结束；而瞬态表中的数据将一直保存到服务器关闭为止。
- 临时表中的数据对于其他会话/连接不可视。
- 由于临时表将很少执行检查工作来了解是否存在并行性冲突，因此临时表比瞬态表的运行速度更快。

如果您不想“共享”您的数据并且与当前会话断开连接之后也不需要这些数据，那么请选择临时表，因为它们的开销很小并且性能更高。

索引

如果一个表存储在内存中，那么有关此表的所有索引也存储在内存中。当然，这将提高性能，但是也会消耗内存空间。

通常，内存索引可以非常快，建议您使用它们来确保快速访问表中的数据。

如果您没有足够的内存，无法将所有表和索引都存储在内存中，那么在某些情况下添加特定索引可能不是最佳选择，这是因为尽管这样做将提高某些查询的速度，但是它也会消耗一些内存，从而会降低其他查询的速度，否则可以使用这些内存来放置其他表。

4 配置内存数据库

本章描述如何配置 solidDB 内存数据库以满足您的环境、性能和操作需要。此描述是《IBM solidDB 管理员指南》的『配置 solidDB』这一节中包含的一部分信息。

请参阅第 39 页的附录 C，『配置参数』，以获取关于与 solidDB 主内存引擎相关的可用参数的完整列表的概述。

配置文件和参数设置

solidDB 从 solid.ini 文件中获取它的大多数配置信息。更确切地说，其实有两个不同的 solid.ini 配置文件，一个位于服务器上，另一个位于客户端。这两个配置文件都不是必需的。如果没有配置文件，那么将使用出厂值。这两个 solid.ini 配置文件中分别包含客户端和服务器的配置参数。如果使用了 ODBC 驱动程序，那么将使用客户端配置文件，并且此文件必须位于应用程序的工作目录中。

注: 在 solidDB 文档中，引用的 solid.ini 文件通常是指服务器端的 solid.ini 文件。

当 solidDB 启动时，它首先将尝试从 SOLIDDIR 环境变量设置的目录中打开 solid.ini 文件。如果在由此变量指定的路径中未找到此文件或者未设置此变量，那么服务器或客户端将尝试从当前工作目录中打开此文件。(当前工作目录通常就是您从其中启动 solidDB 服务器或者客户机应用程序的目录。) 您可以通过使用 -c 服务器命令行选项来指定另外的工作目录。有关命令行选项的更多信息，请参阅《IBM solidDB 管理员指南》中的『附录 B，solidDB 命令行选项』。

配置文件中包含 solidDB 参数的设置。如果在 solid.ini 文件中未设置特定参数的值，那么 solidDB 将使用此参数的出厂值。出厂值可能取决于您使用的操作系统。

通常，出厂值提供了良好的性能和可操作性，但是在某些情况下，修改某些参数值可以提高性能。

可以通过在 solid.ini 文件中设置参数名/值对来修改配置。例如，要指定服务器的网络地址，可使用参数名 Listen 和适当的值，例如，

```
Listen=tcp 192.168.255.1 1315
```

这表示当服务器侦听客户机请求时，它应当使用 TCP/IP 协议、网络地址 192.168.255.1 和端口号 1315 来侦听。

这些参数是按照此配置文件中的节类别进行分组的。请参阅《IBM solidDB 管理员指南》中的『附录 A，服务器端配置参数』和『附录 B，客户端配置参数』，以获取关于节类别和所有可用参数的概述。

每个节类别都以一个用方括号括起来的节名开头，例如:

```
[com]
```

[com] 这一节将列示通信信息。请注意，节名不区分大小写。节名“[COM]”、“[Com]”和“[com]”是等价的。

下面是服务器端的 solid.ini 配置文件的一个样本节:

```
[IndexFile]
FileSpec_1=C:\soldb\solid1.db 1000M
CacheSize=64M
```

管理服务器端的参数

可以通过下列方式来查看和修改 solidDB 参数和它们的值:

- 在 solidDB SQL 编辑器（teletype 工具）中输入下列命令:

  ADMIN COMMAND 'parameter'

  和

  ADMIN COMMAND 'describe parameter'

- 直接编辑 solidDB 目录中的 solid.ini 文件。

后续章节包含有关管理 ADMIN COMMAND 和 solid.ini 的参数的指示信息。

注: 有关仅查看和设置服务器通信协议参数的详细信息，请参阅《IBM solidDB 管理员指南》中的『管理网络连接』这一节。

使用 ADMIN COMMAND 查看和设置参数

借助 ADMIN COMMAND，可以通过 solidDB 服务器来远程更改参数，并且无需重新启动服务器。所有参数都可访问，即使它们在 solid.ini 配置文件中不存在也是如此。如果参数不存在，那么将使用出厂值。

查看参数

可以通过执行以下命令获得一个参数或许多参数的总结视图:

```
ADMIN COMMAND 'parameter [-r] [section_name[.parameter_name]]';
```

其中:

- -r 选项表示只有当前值是必需的
- *section_name* 是此参数在 solid.ini 中所属类别的名称

要查看所有参数，请在 solidDB SQL 编辑器（teletype 工具）中输入以下命令:

ADMIN COMMAND 'parameter';

将返回具有*当前值*、*缺省值*和*出厂值*的所有参数的列表。可以通过添加节名将查看的参数仅限于特定节，例如:

```
ADMIN COMMAND 'parameter logging';
```

可以通过给定完整参数名来查看与单个参数相关的值，例如:

```
admin command 'parameter logging.durabilitylevel';
    RC TEXT
    -- ----
    0 Logging DurabilityLevel 3 2 2
1 rows fetched.
```

将按以下顺序显示上述三个值:

- *当前值*
- 当服务器启动时所使用的*启动值*
- 产品中预先设置的*出厂值*

如果需要，还可以使用 -r 选项来限定此命令以便仅显示当前值。例如:

ADMIN COMMAND ’parameter -r’;

查看对特定参数的描述

还可以查看对特定参数的更详细描述，其中包括有效参数类型和访问方式。这是很有用的信息，特别是因为可能需要动态处理参数；不同的产品、平台或发行版可能具有不同的参数支持。

要查看对某个参数的描述，请使用 solidDB SQL 编辑器（teletype 工具）输入以下命令:

```
ADMIN COMMAND 'describe parameter [section_name[.parameter_name]] ';
```

单个参数的结果集看起来为如下所示:

```
admin command 'describe parameter logging.durabilitylevel';
    RC TEXT
    -- ----
    0 DurabilityLevel
    0 Default transaction durability level
    0 LONG
    0 RW
    0 2
    0 3
    0 2
7 rows fetched.
```

结果集的各行分别是:

- *参数名*是参数的名称，例如 CacheSize。
- 对此参数的*描述*
- *数据类型*
-

访问方式，它可以是下列其中一项:

–

RO: 只读，不能动态更改值

–

RW: 读/写，可以动态更改值，并且更改将立即生效

–

RW/STARTUP: 可以动态更改值，但是要在服务器下一次启动之后更改才生效。

–

RW/CREATE: 可以动态更改值，但是要在创建新的数据库时更改才生效。

•

*启动值*显示参数的启动值

•

*当前值*显示参数的当前值

•

*出厂值*显示产品中预先设置的值。

设置参数值

要设置特定参数的值，请使用 solidDB SQL 编辑器（teletype 工具）输入以下命令:

```
ADMIN COMMAND 'parameter section_name.parameter_name=value [temporary]';
```

其中:

value 是有效参数值。

注:

如果未指定任何值，那么此命令将为此参数设置出厂值（或者取消设置值）。而且，如果您指定一个带有星号（*）的参数值，那么此参数将设置为它的出厂值。

当设置为 temporary 时，不会将已更改的值存储在 solid.ini 文件中。

请注意，您可以在等号两边添加空格。

示例:

```
--set communication trace on
ADMIN COMMAND 'parameter com.trace = yes';
```

注:

参数管理操作并不是事务的一部分，无法回滚。

命令将返回新值作为结果集。如果此参数的访问方式为 RO（只读），或者输入的值无效，那么 ADMIN COMMAND 语句将返回一个错误。

对参数所作修改的持久性

对具有 RW* 访问方式的参数所作的所有更改都存储在下一个检查点的 solid.ini 文件中。这不适用于使用临时选项设置的值。

还可以使用以下命令来请求立即存储已更改的值:

```
ADMIN COMMAND 'save parameters [ini_file_name]';
```

当未指定 *ini_file_name* 时，将重新写入当前 solid.ini 文件。否则，会将完整的配置文件写入一个新位置。这样可以很方便地保存配置文件检查点供以后使用。

在 solid.ini 中查看和设置参数

1. 打开位于 solidDB 进程的工作目录中的 solid.ini 文件。
2. 查看参数的值。

   所显示的参数是当前在服务器中处于活动状态的参数。如果尚未设置参数值，那么启动时将使用出厂值。出厂值可能取决于 solidDB 所运行于的操作系统。
3. 必要时，请添加节、参数和参数值。
4. 保存更改。

必须重新启动服务器才能激活所作的更改。

常量参数值

配置文件的 [IndexFile] 节中的 Blocksize 参数的参数访问方式为 RO。此参数是在创建数据库时设置的，以后不能修改。

如果您要使用其他常量值，那么必须创建新的数据库。在创建新的数据库之前，通过编辑 solidDB 目录中的 solid.ini 文件来设置新的参数常量值。

以下示例将通过向 solid.ini 文件中添加下列各行来为索引文件设置新的块大小:

```
[IndexFile]
Blocksize = 4096
```

在编辑并保存 solid.ini 文件之后，移动或删除旧的数据库和日志文件，然后启动 solidDB。

注:

可以在启动服务器时更改日志块大小。

附录 A. 计算最大 BLOB 大小

用途

内存表与基于磁盘的表之间的一个重要差别是，内存表中的列值必须放入单个“页”中（页大小是在 solid.++ini 配置文件中指定的，其最大值为 32KB）。因此，内存表不能存储超过页大小的字符文件或二进制文件。但是，较小的二进制文件还是受支持的。

本附录说明了如何计算将放入内存表中的字符或二进制列值的最大大小。

背景

目前，许多应用程序使用的数据是不能很容易使用 INT 和 CHAR 等标准数据类型来存储的数据。然而，使用长整型字符或二进制格式可能更合适。在这些情况下，数据可能是分别作为 *CLOB*、*BLOB*、*字符*和二*进制大对象*进行存储的。 CLOB 中包含可解释的字符，字符数目最高可达两百万。BLOB 数据类型实际上可以存储任何可以作为一系列二进制数（8 位字节）来存储的数据。BLOB 通常用来存储不能很容易地解释为数字或字符的大型可变长度数据。例如，BLOB 可以存储已数字化的音频（例如，CD 上保存的音乐）、多媒体文件或者从传感器读取的时序数据。

在 solidDB 中，BLOB 受到广泛支持，并且有多种不同的数据类型可供选择：BINARY、VARBINARY 和 LONG VARBINARY，其中最新的一种数据类型被映射至标准数据类型 BLOB。

CLOB 是使用 CHAR、WCHAR、VARCHAR、WVARCHAR、LONG VARCHAR 和 LONG WVARCHAR 这六种数据类型实现的。两种最新的数据类型被映射至标准数据类型 CLOB 和 NCLOB。有关 CLOB 和 BLOB 数据类型的详细信息，请参阅《IBM solidDB SQL 指南》的*附录 A* 中的『字符数据类型』和『二进制数据类型』这两节。

对于基于磁盘的表，solidDB 实现的 BLOB 存储器既兼顾了访问速度，同时又满足了能够存储大量数据的需要。无论哪种数据类型（VARCHAR 和 VARBINARY 等），较短的值通常都存储在表中，而较长的值的一部分或所有数据存储在数据库存储器树中的一个单独区域。这对于用户是完全透明的；用户只需决定数据类型，其余任务都由 solidDB 完成。无论数据的实际物理位置在何处，都将以相同方式来访问这些数据并且它们以存储在表中来出现。在基于磁盘的表中，VARCHAR 或 VARBINARY 字段的最大长度为 2 GB。

对于内存表，BLOB 数据完整地存储在表中，BLOB 的最大长度受到“块大小”的限制（内存表的每一行都不能超过一页或一个“块”的长度）。在本附录中，提供了一些信息帮助您估计可以存储在内存表中的 VARCHAR 或 VARBINARY 数据的最大大小。

计算

请注意，用于计算可用于 BLOB 的空间的算法是相似的。生成下表的副本，然后在其中填写适合于您的表的值。遵循这些步骤来计算可用于 BLOB 数据的剩余空间。

表 4. 计算可用于 BLOB 数据的空间

	值	要在值中输入的内容	值的含义
1		在左边的空白处，输入块大小或者 32767 这两者中的较小者。块大小将为您在 [IndexFile] BlockSize solid.ini 配置文件中设置的值，或者是《IBM solidDB 管理员指南》中说明的缺省值。	块大小（页大小）是一个“块”中的字节数，与磁盘块相似。由于每一行都必须适合于一个块，因此，此值表示一行的最大大小。
2	17	使用左边显示的硬编码值。	这是每一页的开销的字节数。
3	10	使用左边显示的硬编码值。	这是每一行的开销的字节数。 如果您有大型 BLOB，那么我们将假定每一页只有一行。
4		如果您已经为表声明了一个显式主键，那么请输入值 10，否则请输入 20。	此值表示用于服务器自动为每个表添加的列的字节数。
5		输入表中的列数再乘以 2 所获得的结果。	这是所有列的开销的字节数。
6		输入表中具有固定大小的数据列的大小的总和。（请参阅下面的表 #2，以了解具有固定大小的每种数据类型的大小。）	这表示具有固定大小的列所占用的空间。
7		输入 BLOB 列的数目。	这是用来终止 BLOB 值的字节数（每个值使用一个字节）。
8		将第 2 行到第 7 行中的值求和。	这是除 BLOB 值之外的每个值使用的空间的总和。
9		用第 1 行的值减去第 8 行的值。	这是大约可用于 BLOB 数据的字节数。如果您的表中只有单个 BLOB 列，那么这是此 BLOB 值的最大大小的近似值。

注:

注意：最大块大小为 64K；但是，最大行大小（从而计算出最大 BLOB 大小）只是 32K（实际上为 32K-1，也就是 32767 个字节）。如果块大小为 64K 或 32K，那么请输入 32767，而不是输入表的第一行中的块大小。

下表指示存储每种具有固定大小的数据类型的值所需要的字节数。例如，它使用 8 个字节来存储一个类型为 SQL FLOAT 的值。

表 5. 存储值所需要的字节数

数据类型	存储器大小（按字节计）
TINYINT	1
SMALLINT	2
INT	4
BIGINT	8
DATE/TIME/TIMESTAMP	11

表 5. 存储值所需要的字节数 (续)

数据类型	存储器大小（按字节计）
FLOAT / DOUBLE PRECISION	8
REAL	4
NUMERIC / DECIMAL	11
CHAR / VARCHAR / LONG VARCHAR	char_length(column_value) + 1
WCHAR / WVARCHAR / LONG WVARCHAR	char_length(column_value) * 2 + 1
BINARY / VARBINARY / LONG VARBINARY	octet_length(column_value) + 1

附录 B. 计算存储空间需求

此附录提供了一些信息使您能够估计将一个表及其索引存储在内存或者磁盘中所需要的内存或磁盘空间量。

请注意，下面所提供的公式可能会因多种原因而不准确，这些原因包括:

- solidDB 压缩了某些数据。
- 根据所存储的值的实际长度不同，可变长度的数据（例如，VARCHAR）需要不同的空间量。
- 内存数据结构并不需要为每条记录存储相同数目的“指针”。

在本讨论中，我们大多数情况下都假定未对数据进行压缩，并且使用了最大数目的指针。因此，您通过使用这些公式获得的结果通常会有点保守，也就是说，此公式通常会过高估计需要的空间量。

在下面的公式中，表示法

```
sum_of(x)
```

意味着求出每个“x”的大小的总和。例如，

```
sum_of(col_size)
```

意味着求出表或索引中每一列的大小的总和，

```
sum_of(index_sizes)
```

意味着求出表的所有索引的大小的总和。

基于磁盘的表

用于计算基于磁盘的表所需要的空间的一般公式为:

```
chkpt_factor x (table_size + sum_of(index_sizes))
其中:
        chkpt_factor 在 1.0 到 3.0 之间（下面对此进行了说明），并且
        table_size =
            1.4 x rows x (sum_of(col_size + 1) + 12)
其中:
        rows 是行数;
        sum_of(col_size + 1) 是所有列的大小加上每列一个字节所获得的总和。
列大小显示在表后面。
```

对于每个基于磁盘的索引，index_size 为

```
1.4 x rows x (pkey_size + idx_size)
```

其中 pkey_size 是主键中各列大小的总和，idx_size 是索引中各列大小的总和。

chkpt_factor 需要考虑“检查点”操作可能最多达数据库大小的三倍。在执行检查点操作期间，会将数据库中每个已更改的页的副本从内存复制到磁盘中。如果已更新数据库中的每一页，那么可以根据磁盘上已经具有的页数来从内存中复制尽量多的页数。而且，在成功完成当前检查点之前，不会删除最新的成功检查点。因此，在完成检查点操作期间，磁盘中最多可以同时具有每一页的 3 个副本（此页在数据库中的一个副本，最新的成功检查点中的一个副本，当前正在执行的检查点的一个副本）。因此，检查点因子可以在 1.0 到 3.0 之间。在大多数数据库中，此值很少接近 3.0。通常，此值为 1.5 就足够了，即使对于具有高级活动的小型数据库也是如此。请注意，检查点的频率越低，chkpt_factor 可能就需要更大。

注:

在基于磁盘的索引中，如果您未显式定义主键，那么服务器将使用由服务器生成的“行号”作为主键。这将强制主键索引按记录的插入顺序来存储记录。

内存表

适用于内存表的一般公式为:

```
table_size + sum_of(index_sizes)
```

table_size =

```
1.3 x rows x (sum_of(col_sizes) + (3 x word_size) + (2 * num_cols) + 2)
```

其中: rows 是行数;

word_size 是机器字大小（例如，对于 32 位操作系统为 4 字节，对于 64 位操作系统为 8 字节）;

num_cols 是列数;

sum_of(*col_sizes*) 是所有列大小的总和。

对于每个内存索引，索引大小为

```
1.3 x rows x ((dist_factor x sum_of(col_sizes + 1)) + (8 x word_size) + 4)
```

其中“dist_factor”是一个在 1.0 到 2.0 之间的值，这取决于键值的分配。如果键值非常不相似，那么请使用一个很接近 2.0 的值。如果键值非常相似，那么请使用一个很接近 1.0 的值。

列大小表

TINYINT: 2 个字节

SMALLINT: 2 个字节

INT: 4 个字节

BIGINT: 8 个字节

DATE/TIME/TIMESTAMP: 11 个字节

FLOAT / DOUBLE PRECISION: 8 个字节

REAL: 4 个字节

NUMERIC / DECIMAL: 12 个字节

CHAR / VARCHAR / LONG VARCHAR: char_length(column_value) + 5

WCHAR / WVARCHAR / LONG WVARCHAR: char_length(column_value) * 2 + 5

BINARY / VARBINARY / LONG VARBINARY: octet_length(column_value) + 5

测量内存消耗量

在创建您的表和索引之后，可以使用以下命令来测量实际消耗的内存量:

ADMIN COMMAND ’info imdbsize’;

此命令将提供内存表和索引总共消耗的内存量（以千字节为单位）。

详细信息

本章包含一些有关如何将数据存储在不同存储器树中的详细信息。如果您愿意更好地了解前面的公式的基础知识，那么您可能会发现此信息很有用。

基于磁盘的表

在基于磁盘的表中，数据和索引存储在一个 B 树中。此树中的每个条目都将消耗一定空间用于标题和数据。

可以使用先前显示的列大小的表来计算实际数据所使用的空间。此表中的值都采用最大长度。可变长度的数据（例如，VARCHAR）或者可压缩的数据可能需要更少的字节数。

此外，在基于磁盘的表中，服务器要求对每列再增加一个字节；此字节被用作长度指示符（也充当空指示符）的一部分。

每一行的标题都使用 12 个字节:

表 6. 标题字节数

字节数	用于...
3 个字节	行标题
3 个字节	表标识
6 个字节	行版本

如果基于磁盘的表中包含索引（而不是主键），那么必须使用相同的准则来单独估计这些索引中条目的大小。一个索引条目包含下列组成部分:

- 在此索引中定义的列
-

此表的主键包含的列

•

行标题（12 个字节）

此外，数据库页中通常有一些空白空间（例如，占 20 - 40%）。这就是用于表和索引的公式中都包含一个倍数 1.4 的原因。

例如：我们有一个基于磁盘的表：

```
CREATE TABLE subscriber (
             id INTEGER NOT NULL PRIMARY KEY,
             name VARCHAR(50),
             salary FLOAT) ;
```

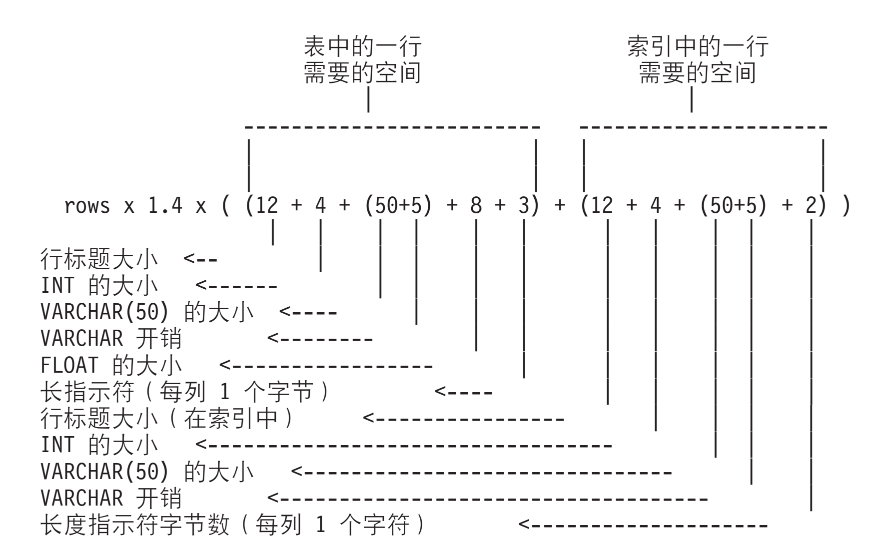
此外，我们创建了一个辅助索引：

```
CREATE INDEX subscriber_idx_name ON subscriber (name);
```

索引条目中包含 NAME 列；它还包含主键列，在此例中为 ID。应单独估计此索引需要的空间。假定“空白空间因子”为 1.4，那么基于磁盘的表的总大小为：

```
rows x 1.4                   // 1.4 = 估计的空白空间大小。
   x ( (12 + 4 + (50+5) + 8 + 3)   // 表条目的大小，
       + (12 + 4 + (50+5) + 2) )   // 辅助索引条目的大小
```

用另一种方式来表示：



内存表

内存表所需要的空间是按不同方法进行估计的。

每个条目的大小是表数据的大小加上每行三个内存指针的开销（在 32 位操作系统中，每个内存指针占用 4 个字节；在 64 位操作系统中，每个内存指针占用 8 个字节）。此外，还应加上每行两个字节以及此行中的每一列两个字节的开销。请注意，每一列不需要为了考虑长度指示符而增加一个字节的开销，因为每行两个字节的开销已经包括此开销。

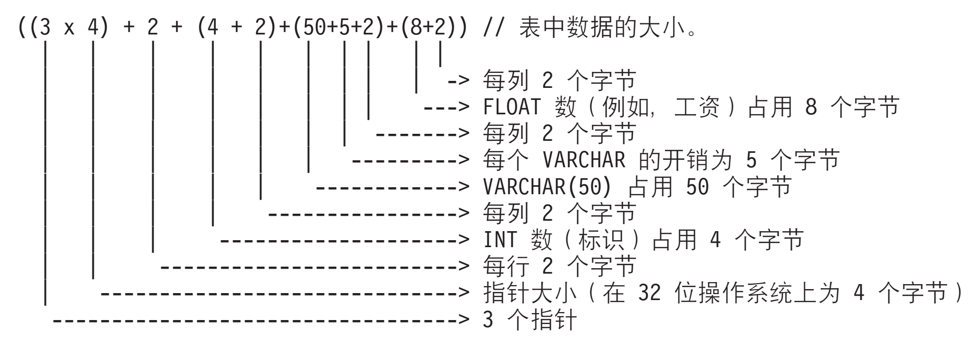
此外，主内存表可能有索引，服务器启动时将填充索引。每个索引条目包含此索引中定义的列的数据。此外，每个索引条目最多可以包含八个内存指针。（请注意，内存索引并不需要主键的副本。）

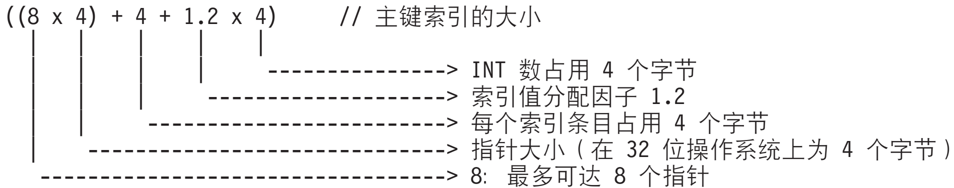
而且，还有一些取决于索引的实际数据值的其他开销。这是索引的数据大小所占的百分比。不能准确地给定准确值，这是因为它取决于键值分配，但是倍数在 1.0 到 2.0 范围内。

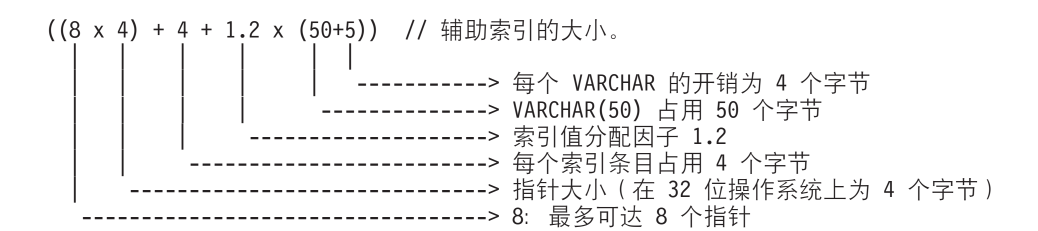
此外，对于索引结构本身，每个索引条目（即，每一行）平均需要 4 个字节。

对于上述示例表和索引，在 32 位操作系统中的内存消耗量可以按如下所示进行估计:

```
rows * 1.3 x (
  ((3 x 4) + 2 + (4 + 2) + (50+5+2) + (8+2)) // 表中数据的大小
+ ((8 x 4) + 4 + 1.2 x 4)       // 主键索引的大小
+ ((8 x 4) + 4 + 1.2 x (50+5)))  // 辅助索引的大小。
)
```







在 64 位操作系统中，内存指针大小为 8 个字节而不是 4 个字节。

在以上估计值中使用的因子 1.2 是"TRIE 索引值分配因子"，其准确值取决于已建立索引的列的实际值。它的值通常在 1 到 2 之间。对于随机值分配，此值更接近 2.0；而对于序列值分配，此值更接近 1.0。一个索引条目平均需要的数据开销为 4 个字节。

使用因子 1.3 时将考虑内存分配器的内部开销。

注:

主内存表的索引是在服务器每次启动时动态创建的；它们决不会写入磁盘中，因此它们不会占用任何磁盘空间。但是，当服务器关闭和执行检查点操作期间，主内存表本身将被写入磁盘。因此，您必须具有足够大的磁盘空间总量用来同时存储基于磁盘的表和内存表。

附录 C. 配置参数

这些参数是按照 solid.ini 配置文件中的节类别进行分组的。与 solidDB 内存数据库相关的参数被存储到此配置文件的 [MME] 节中。另外一个参数 DefaultStoreIsMemory 被存储到 [General] 节中。

可以通过手动编辑 solid.ini 配置文件或者通过在 solidDB SQL 编辑器中输入以下命令来更改配置参数:

```
ADMIN COMMAND 'parameter section_name.param_name=value'
```

例如:

ADMIN COMMAND ’parameter mme.imdbmemorylimit=1gb’;

注:

服务器仅在它启动时才读取此配置文件，因此，对此配置文件所作的更改将在服务器下一次启动之后才会生效。

下面提供了与 IMDB 相关的配置参数的完整列表。

[General] 节

表 7. MME 参数

[General]	描述	出厂值	访问方式
DefaultStoreIsMemory	如果此参数设置为 Yes，并且在 CREATE TABLE 语句中不包含显式 STORE 子句的情况下创建新表，就会作为内存表来创建这些新表。如果此参数设置为 NO，那么缺省情况下会将新表存储在磁盘上。可以通过在 CREATE TABLE 语句中使用 STORE 子句来覆盖缺省值。 此参数仅适用于支持 solidDB 内存数据库的产品。 请注意，即使此参数设置为 Yes，系统表也是存储在磁盘上。	No	RW

[MME] 节

注:

（solid.ini 文件 [General] 节中的）DefaultStoreIsMemory 参数也与 solidDB 内存数据库相关。有关更多信息，请参阅第 39 页的『[General] 节』。

表 8. MME 参数

[MME]	描述	出厂值	访问方式

ImdbMemoryLimit	此参数设置一个服务器将为内存表及其索引分配的内存（虚拟内存）量的上限。请注意，“内存表”包括临时表和瞬态表以及“正常的”（持久）内存表。
	
	可以按字节、千字节（KB）、兆字节（MB）或吉字节（GB）来指定内存限制。例如:
	
	`ImdbMemoryLimit=1073741824`
	`ImdbMemoryLimit=1048576kb`
	`ImdbMemoryLimit=1024MB`
	`ImdbMemoryLimit=1GB`
	
	如果将它的值设置为 0，那么表示“无限制”。
	
	一般来说，对于内存不超过 1GB 的服务器，应为内存表分配的最大内存量通常是系统的物理内存的 30% 到 70%。系统的内存越大，可以用于内存表的内存所占的百分比就越大。
	
	注意：此参数仅适用于 solidDB 主内存引擎表。它不适用于其他版本的 solidDB 或基于磁盘的表。
	
	可以使用以下命令来更改此内存量:
	
	`ADMIN COMMAND 'parameter MME.ImdbMemoryLimit=n[kb|mb|gb]';`
	
	其他“n”是一个正整数。当服务器正在运行时，您只能增大而不能减小此值。此命令将立即生效。在服务器关闭时，新值将被写入 solid.ini 文件。
	
	注意:
	
	强烈建议您确保内存表满足可用物理内存的要求。如果超过了可用的物理内存量，那么性能将显著下降。如果您用尽了所有可用的虚拟内存，那么服务器将突然限制执行插入和更新等操作并且返回错误代码。
		0
		
		单位：1 字节 k=KB m=MB g=GB
			RW

表 8. MME 参数 (续)

[MME]	描述	出厂值	访问方式
ImdbMemoryLowPercentage	一旦您设置了 ImdbMemoryLimit，那么可以设置此附加参数，以便在用尽所有内存之前提前对您发出警告。此 ImdbMemoryLowPercentage 参数允许您指出在服务器开始限制您执行将行插入内存表等操作之前可以使用的内存量所占的百分比。例如，如果 ImdbMemoryLimit 为 1000MB，并且 ImdbMemoryLowPercentage 为 90（百分比），那么当内存表用完了 900 MB 的内存时，服务器将停止接受插入操作。 有效值为 60 到 99 之间的数（百分比）。 **注:** 此参数仅适用于 solidDB 主内存引擎表。它不适用于其他版本的 solidDB 或基于磁盘的表。	90	RW/Startup
ImdbMemoryWarningPercentage	此参数为 IMDB 内存大小设置一个警告限制。此警告限制用 ImdbMemoryLimit 参数值的百分比来表示。超过 ImdbMemoryWarningPercentage 限制时，就会产生一个系统事件。 将自动检查 ImdbMemoryWarningPercentage 参数值的一致性。它必须小于 ImdbMemoryLimit 参数值。 **注:** 此参数仅适用于 solidDB 主内存引擎表。它不适用于其他版本的 solidDB 或基于磁盘的表。	90	RW/Startup
LockEscalationEnabled	通常，当服务器需要使用锁定来防止发生并行性冲突时，服务器将锁定各个行。这意味着每个用户将只影响其他想使用相同行的用户。但是，锁定的行越多，服务器就必须花更多时间来检查是否存在相冲突的锁定。在某些情况下，更值得锁定整个表而不锁定此表中的许多行。 当 LockEscalationEnabled 设置为 yes 时，在当前事务中锁定了同一个表中的指定行数之后，锁定级别将从行级别升级为表级别。通过锁定升级虽然提高了性能，但是降低了并行性，因为其他用户将暂时不能使用同一个表，即使他们想使用此表中的其他行也不行。请参阅 LockEscalationLimit 参数。 此参数的值可以为“yes”或“no”。 **注:** 此参数仅适用于内存表。	yes	RW/Startup

表 8. MME 参数 (续)

[MME]	描述	出厂值	访问方式
LockEscalationLimit	如果 LockEscalationEnabled 设置为 yes，那么此参数指示必须锁定单个表中的多少行之后，服务器才会将锁定级别从行级别升级为表级别。（请参阅 LockEscalationEnabled 以了解更多详细信息。） 值可以使用 1 到 2,147,483,647（2^32-1）之间的任何数。 **注:** 此参数仅适用于内存表。	1000	RW/Startup
LockHashSize	服务器使用散列表（数组）来存储锁定信息。如果明显低估了数组大小，那么性能将下降。尽管太大的散列表会产生内存开销，但是它们并不会直接影响性能。LockHashSize 用于确定散列表中的元素数。 当服务器使用悲观并行控制（即，锁定）时，需要提供此信息。服务器对内存表和基于磁盘的表使用不同的数组。此参数适用于内存表。 通常，需要的锁定越多，此数组就应该越大。但是，很难计算所需要的锁定数，因此，您可能需要通过试验来找到最适合于您的应用程序的值。 您输入的值是散列表条目数。每个表条目的大小就是一个指针的大小（在 32 位体系结构中为 4 个字节）。因此，假若您选择一个散列表的大小为 1,000,000，那么需要的内存量为 4,000,000 个字节（假定是 32 位指针）。	1000000	RW/Startup
MaxCacheUsage	在对 M 表执行检查点操作时，MaxCacheUsage 的值将限制使用 D 表高速缓存的数量。预计此值是按字节给定的。无论 MaxCacheUsage 是何值，最多只能将 D 表高速缓存（IndexFile.CacheSize）的一半用于对 M 表执行检查点操作。值 MaxCacheUsage=0 将设置不受限制的值，这意味着使用的高速缓存为 IndexFile.CacheSize/2。	8MB	RW/Startup
ReleaseMemoryAtShutdown	当设置为“yes”时，将要求服务器在关闭时应当显式释放内存表所使用的内存，而不应依赖操作系统来清理与此过程相关联的所有内存。某些操作系统（例如 VxWorks）可能要求您将此参数设置为“yes”，以确保释放所有内存。 值可以是 yes 和 no。 出厂值是 no，因为这种情况下将更快关闭服务器。	No	RW/Startup

索引

[B]

表
- 持久内存表 1
- 非持久内存表 2
- 局限性 11
- 临时 2，6，13，20
- 内存 1，4，11，21
- 内存表类型 1
- 瞬态 2，6，15，20
- 索引 21
- 指定 4

[C]

参数
- 达到限制 7
- Blocksize 27
- CacheSize 6
- DefaultStoreIsMemory 4
- ImdbLowPercentage 6，7
- ImdbMemoryLimit 6，7
- ImdbMemoryWarningPercentage 6，7
- ProcessMemoryCheckInterval 6，10
- ProcessMemoryLimit 6，9
- ProcessMemoryLowPercentage 6，10
- ProcessMemoryWarningPercentage 6，10

存储空间需求
- 对于基于磁盘的表 33
- 对于内存表 34
- 计算 33，34

[D]

等号
- 设置参数值时使用 39

[K]

可重复读 11
可序列化 11
- 使用限制 11

[L]

链接库访问 12
临时表
- 持续时间 14
- 局限性 14
- 可视性 14

临时表 *(续)*
- 无法用作主控表 14
- 与 ImdbMemoryLimit 的关系 6
- 在具有引用约束的情况下使用 14

落实读 11

[N]

内存
- 物理 11
- 虚拟 11

内存表的局限性 11
内存消耗量
- 测量 35
- 监控 5
- 控制 4，6

[P]

配置
- 参数设置 23
- 查看参数 24
- 查看参数描述 25
- 出厂值 23
- 服务器端配置文件 23
- 管理参数 24，25，26
- 客户端配置文件 23
- 内存数据库 23
- 配置文件 23
- 缺省设置 23
- 设置参数 24，26
- 示例 23
- solid.ini 23

配置文件
- 在服务器上 23
- 在客户端上 23

[S]

事务隔离 11
事务隔离级别
- 可重复读 11
- 可序列化 11
- 落实读 11
- 使用限制 11

数据库
- 表 1
- 表类型 1
- 持久表 1
- 磁盘空间需求 10

数据库 *(续)*
非持久表 2
更改表类型 11
临时表 2, 13, 20
内存 1, 2, 3, 11, 19, 23, 24
内存表提高性能 3
配置 23, 24
瞬态表 2, 15, 20
性能调整 20
要选择的表 3, 19
瞬态表
持续时间 15
局限性 15
无法用作主控表 15
与 ImdbMemoryLimit 的关系 6
在具有引用约束的情况下使用 15
索引
在内存表中 21

[Y]

用于选择要存储在内存中的表的算法 19

A

ADMIN COMMAND
info imdbsize 5
pmon mme 5

B

BLOB
计算最大大小 29
Blocksize（参数） 27

C

CacheSize（参数） 6
CLOB 29

D

DefaultStoreIsMemory（参数） 4

H

HotStandby 11

I

ImdbLowPercentage（参数） 6, 7
ImdbMemoryLimit（参数） 6, 7, 40
ImdbMemoryLowPercentage（参数） 41
ImdbMemoryWarningPercentage（参数） 6, 7, 41
info imdbsize
ADMIN COMMAND 5

L

LockEscalationEnabled（参数） 41
LockEscalationLimit（参数） 42
LockHashSize（参数） 42

M

MaxCacheUsage（参数） 42

P

pmon mme
ADMIN COMMAND 5
ProcessMemoryCheckInterval（参数） 6, 10
ProcessMemoryLimit（参数） 6, 9
ProcessMemoryLowPercentage（参数） 6, 10
ProcessMemoryWarningPercentage（参数） 6, 10

R

ReleaseMemoryAtShutdown（参数） 42

S

solid.ini
[MME] 节 6
[SRV] 节 6

[特别字符]

=
设置参数值时使用等号 39

声明

本产品受美国专利 6144941、7136912、6970876、7139775、6978396 和 7266702 的保护。

为此产品指定的美国出口管制分类编号是 ECCN=5D992b。

本信息是为在美国提供的产品和服务编写的。

IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区：INTERNATIONAL BUSINESS MACHINES CORPORATION"按现状"提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的：（i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系：

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息包含在日常业务操作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名字都是虚构的，若现实生活中实际业务企业使用的名字和地址与此相似，纯属巧合。

版权许可：

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。

凡这些实例程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明：

©（贵公司的名称）（年）。此部分代码是根据 IBM Corp. 公司的样本程序衍生出来的。

商标

IBM、IBM 徽标、ibm.com®、Solid®、solidDB、InfoSphere™、DB2®、Informix® 和 WebSphere® 是 International Business Machines Corporation 在美国和/或其他国家或地区的商标或注册商标。如果这些商标和其他 IBM 注册商标在本资料中第一次出现时标有商标符号（® 或 ™），那么这些符号表示它们是发布本资料时归 IBM 所有的经过美国政府注册的商标或普通法商标。这些商标也可能是在其他国家或地区的注册商标或普通法商标。在 Web 上的版权和商标信息（www.ibm.com/legal/copytrade.shtml）处提供了 IBM 商标的最新列表。

Java™ 和所有基于 Java 的商标和徽标是 Sun Microsystems, Inc. 在美国和/或其他国家或地区的商标。

Linux® 是 Linus Torvalds 在美国和/或其他国家或地区的注册商标。

Microsoft and Windows 和 Microsoft Corporation 在美国和/或其他国家或地区的注册商标。

UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

IBM

中国印刷

S151-1146-00

